1. Thomas : Intro storytelling Thomas sur Alister et J. Chassaing
    1. Thomas fini en ouvrant sur quelques questions mais sans y répondre :
        1. Qu'est ce que c'est que cette alternative ?
        2. Peut-on encore parler d'archi hexagonale ?
        3. Quels sont les avantages/inconvénients ?
        4. Quand est-il utile de l'utiliser ?
2. On se présente : Thomas puis Bruno
3. Bruno : Présentation du déroulé :
    1. Rappel de ce qu'est l'archi hexagonale, puis présentation de ce qu'est le functional core
    2. On va partir d'une code base en archi hexagonale et on va la transformer ensemble en functional core / imperative shell
    3. On va finir en comparant ce pattern avec l'archi hexa
4. Bruno qui présente / Thomas qui commente, questionne, précise : **Partie théorique** :
    1. Archi hexagonale (slides)
    2. Functional core / imperative shell (slides)
5. Thomas qui manipule l'IDE / Bruno qui commente, questionne, précise : **Partie pratique**
    1. Présentation de la solution et de l'hexagone (focus sur les tests, et le web controlleur)
    2. On sort les effets de bords de notre hexagone
    3. On constate que les méthodes de nos controlleurs web sont des imperative shells, mais
    4. Remarques de Bruno :
        1. c'est facile car notre code métier était déjà immutable (et oui, on fait du DDD et on utilise des value types dès que possible)
        2. Notre imperative shell n'est pas super orienté fonctionnel. On peut faire mieux avec le functor Maybe()
6. Bruno qui récupère la main et le code / Thomas qui commente, questionne, précise :
    1. Introduction de la Maybe : ce que c'est, pourquoi il est utile de l'utiliser
    2. Copier-colle de la Maybe depuis le site de Mark Seeman
    3. Mise en place. Explication de ce qui se passe en termes de cinématique d'appels (pour celles et ceux qui ne font pas de C#)
    4. Discussion de son intérêt lorsqu'il y a encore plus d'appel à des fonctions de notre core (on change de modèle : au lieu d'avoir un appel unique à notre domaine, on chaine les appels de différentes fonctions qui appartiennent à notre domaine
    5. Revue du diagramme de Thomas avec étapes de workflow d'appels
7. Discussion :
    1. Ce qui ne semble pas être une option, par exemple en Haskell, peut le devenir dans vos APIs en java, C#, Kotlin etc.
    2. Thomas : Attention, point important : le Domaine fuit légèrement la couche d'infra car le séquencement des appels de fonctions se fait depuis l'extérieur du functional core
    3. Mon heuristique est donc de ne l'utiliser que lorsqu'il n'y a pas trop d'orchestration d'I/Os à faire
    4. Bruno : ça ne peut fonctionner que si votre domaine est codé en style fonctionnel. Ici, notre logique métier n'avait pas d'état ; l'état était injecté au début, au moment de traiter chaque requête
    5. Matrice comparative
8. Conclusion:
    1. Essayez de ne pas rester uniquement dans les modèles que vous connaissez. Creusez, allez voir ce qui existe à côté, mais ne choisissez pas un modèle par mode, sans être

capables de justifier son usage dans votre contexte. Soyez capable d'expliciter vos choix de design.

——

- Objectif : bien séparer la logique métier du reste du code
- Solution élégante : l'architecture hexagonale. C'est quoi (rappel)
- Story telling : a la fin du live coding, Jeremy Chassaing est venu voir Alister pour savoir ce qu'il pensait du pattern fonctional core/
- Auj, on voudrait vous montrer

Un truc qui m'a longtemps obsédé dans ma carrière, c'était de ne pas arriver à bien séparer le code métier du reste, le code plus technique, lié aux librairies ou aux pratiques à la mode etc. On essayait de faire des couches pour bien séparer en fonction d'abstraction : data access, domain, application/use case, IHM etc. Mais chaque concept métier se retrouvait au final un peu partout, et la logique métier également (genre des vérifications de valeurs possibles etc).

Le jour où j'ai découvert l'archi hexagonale, en 2008 ou qque chose comme ça (je suis nul en date)

Call for action:
- Essayez de ne pas rester uniquement dans les modèles que vous connaissez. Creusez, allez voir ce qui existe à côté, mais ne choisissez pas un modèle par mode, sans être capables de justifier son usage dans votre contexte